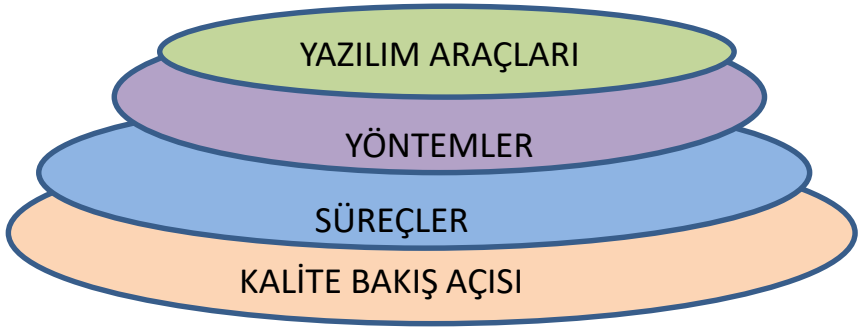
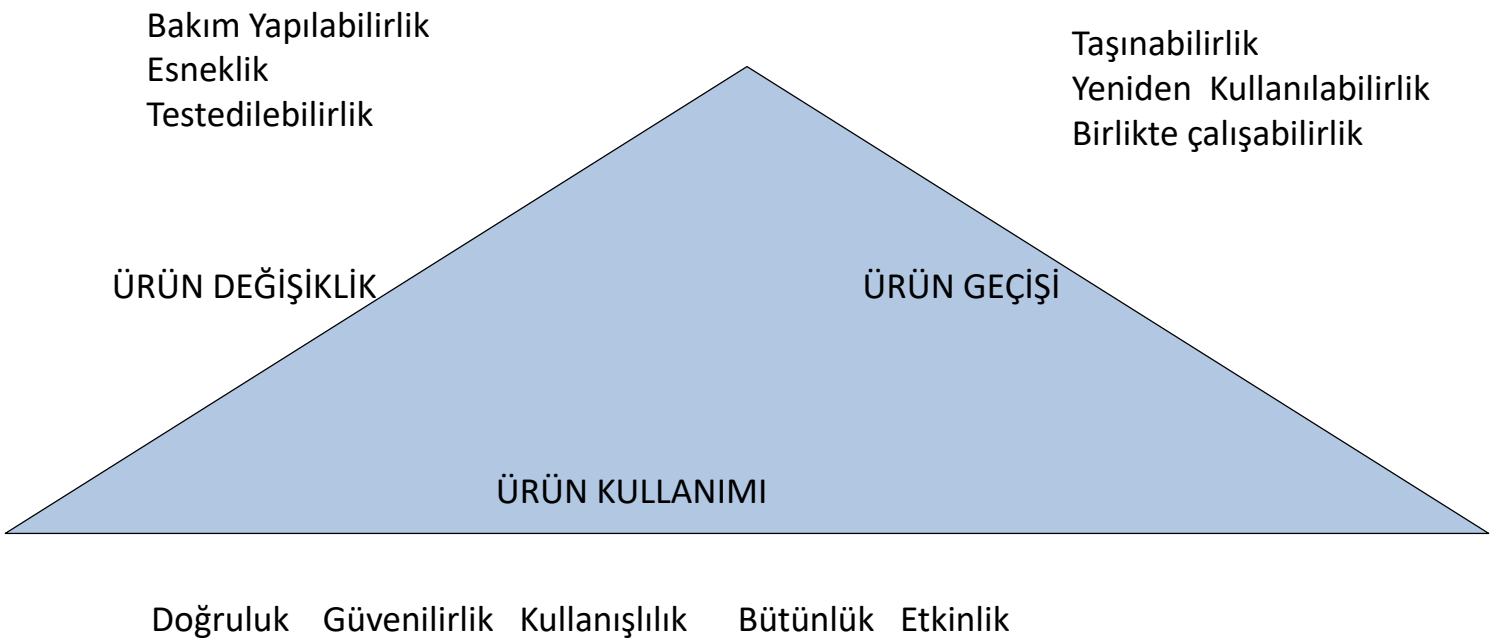


Şekil 1.1. Yazılım Mühendisliği Katmanlı Yapısı

Şekil 2.1: Yazılım Kalite Faktörleri



Şekil 1.1: Mc Call Kalite Üçgeni

Yazılım Kalite Özellikleri (1)

- Doğruluk *(correctness)*: Spesifikasyonlara uygunluk ve müşteri isteklerini karşılama derecesi
- Güvenirlik*(reliability)*: Tasarlanan işlevleri istenilen duyarlıkta yerine getirme olanağı
- Verimlilik *(efficiency)*: Programın işlevlerini yerine getirebilmesi için gerekli bilgi işlem kaynaklarının (algoritmanın gerektirdiği bellek miktarı, işlem süresi) ve kodlamanın gideridir. Bir yazılım sistemi eğer maliyeti düşükse de etkindir.
- Güvenlik *(security)*: Yetkisiz kişilerin yazılıma ve veriye girişini önleme olanağı,
- Kullanışlılık *(usability)* : Öğrenme, işletme , girdi hazırlama ve çıktı yorumlamada kolaylık derecesi

Yazılım Kalite Özellikleri (2)

- Hata bulma Kolaylığı: Hatanın yerini bulma ve düzeltme olanağı
- Esneklik *(flexibility)*: Yazılımda değişiklik yapma kolaylığı
- Sınama Yapılabilirlik *(testability)*: Yazılımım doğruluğunu sınanmada kolaylık
- Taşınabilirlik *(portability)*: Yazılımın farklı donanımlarda ve işletim yazılımı üzerinde kullanılma olanağı, Eğer bir yazılım sistemi başka bir donanım veya yazılım ortamına düşük maliyetle tanışabiliyorsa taşınabilirlik özelliği vardır

Yazılım Kalite Özellikleri (3)

- Yeniden kullanılabilirlik *(reusability)*: Yazılımın tümünün veya bir bölümünün başka bir uygulamada kullanılma olanağı
- Bağlanabilirlik*(interoperability)*: Bir sistemin diğerine bağlanabilme olanağı

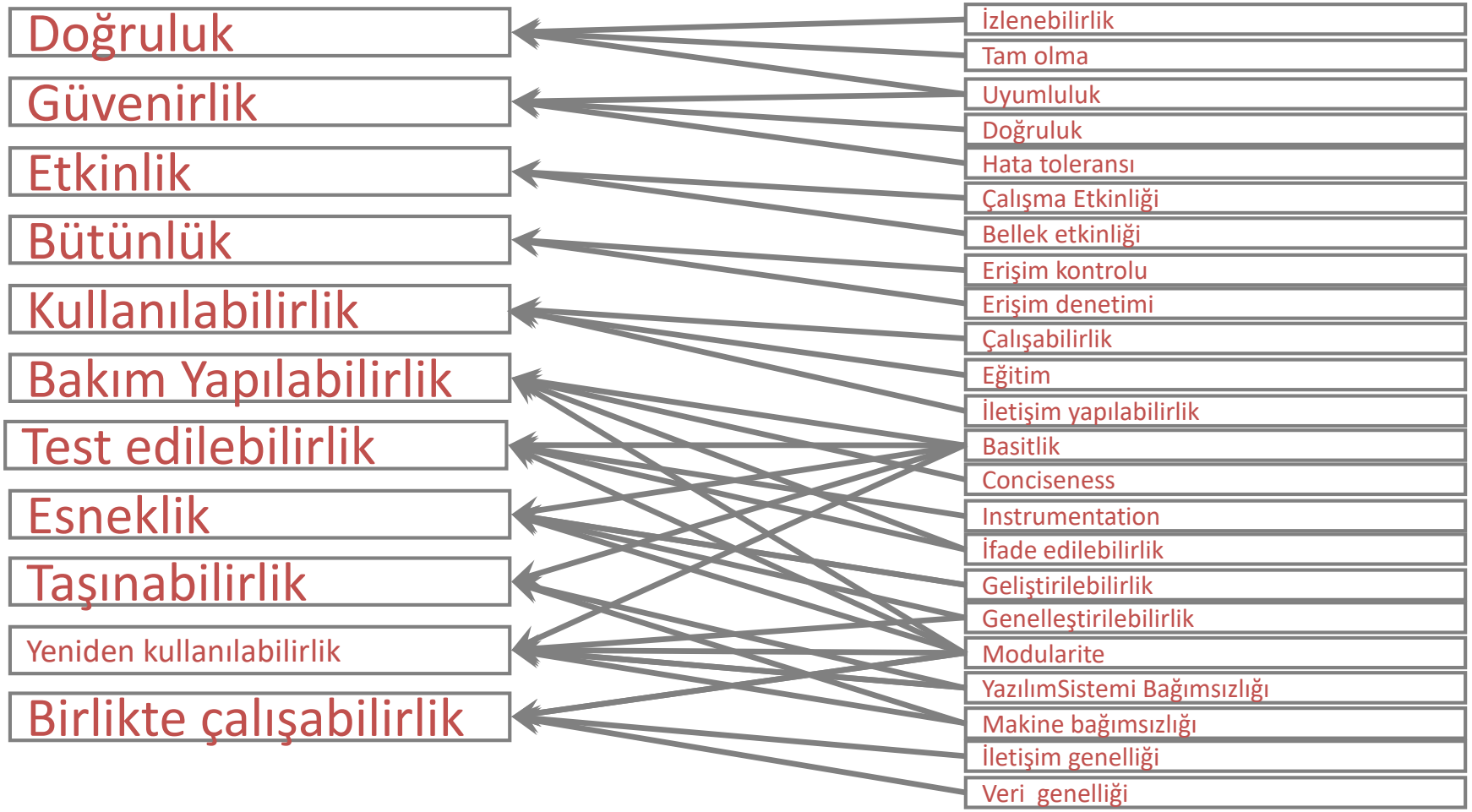
Mc Call Yazılım Kalite Modeli

Yazılım kalite faktörleri (F_i) doğrudan ölçülmeyip, ancak bazı özelliklere (m_j) bağlı birer doğrusal regresyon denklemi;

$$\mathbf{F_i = c_1 m_1 + c_2 m_2 + c_3 m_3}$$

biçiminde kestirilebilir (Şekil 2.2).

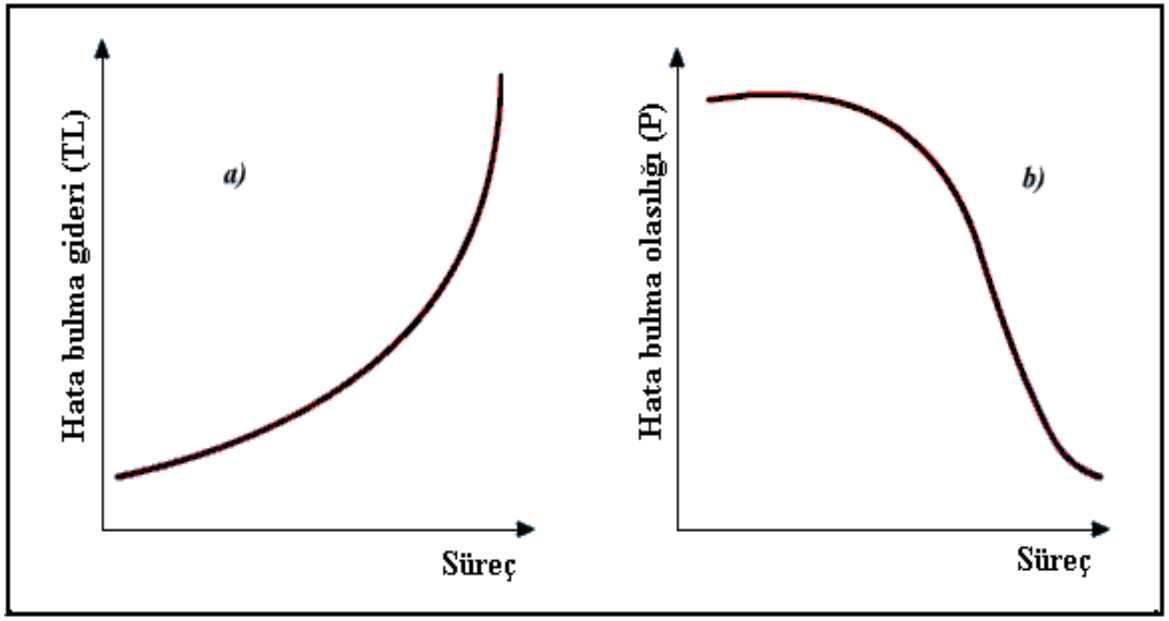
Şekil 2.2: Mc Call Kalite Modeli



Yazılım Kalite Sağlama Aktiviteleri

- Standartlar
- Geçerleme ve Doğrulama (Verification / Validation)
- Gözden geçirme ve denetim (Reviews and Audits)
- Test
- Hata / kusur analizi (error/defect)
- Değişim yönetimi, konfigürasyon yönetimi
- Eğitim
- Risk yönetimi

Şekil. 2.3.Yazılım geliştirme süreci ile hata bulma gideri ve hata bulma olasılığı ilişkisi



2.2.3.Yazılım Mühendisliği Standartları

Yazılım mühendisliği standartları:

- yazılım geliştirme aşamasına,
- kaynağına ve
- değerlendirme ölçütünün tipine

göre üç ayrı biçimde sınıflandırılmaktadır

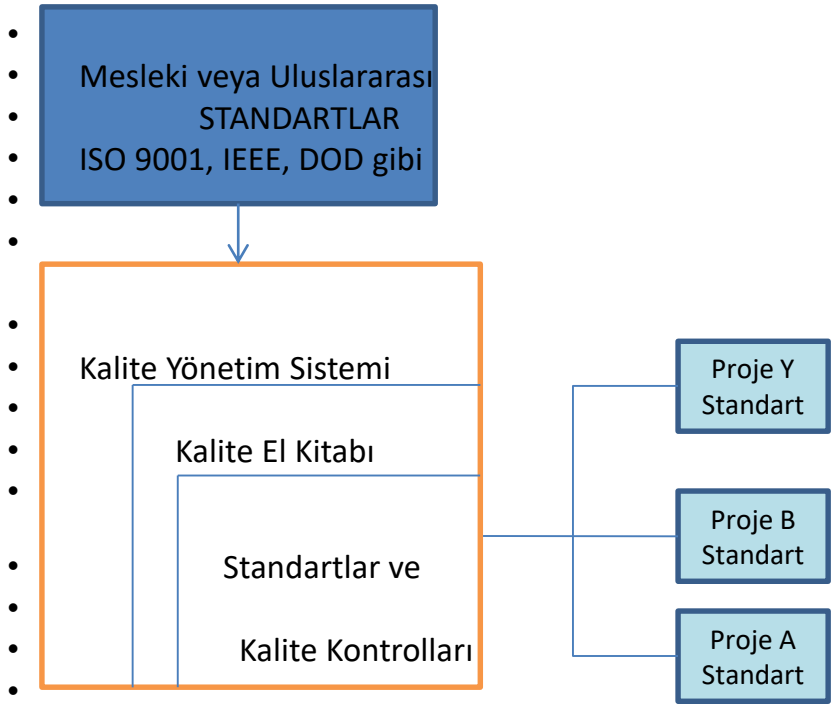
Yazılım geliştirme aşamasına göre Standartları sınıflamada:

- Proje Yönetimi,
- Gereksinim Tanımı,
- Tasarım,
- Kodlama,
- Güvenlik ve Geçerlilik,
- Konfigürasyon Yönetimi,
- Kalite Güvencesi,

gibi standardın etkiliği olduğu yazılım geliştirme süreci adımı belirleyicidir.

3.1.Yazılım Kalite Yönetim Sistemi

- Kalite yönetim sistemi, yazılım ürünlerinin istenen kalite faktörlerine sahip olmasını sağlayan, geliştirici ve kullanıcının birlikte belirlediği yönetim yapısı – sorumluluklar – etkinlikler – yetenekler ve kaynaklardan oluşmaktadır.
- Standartlar ile kalite yönetim sistemi ve projeler arasındaki ilişki, Şekil 3.1 de gösterilmiştir.



Şekil 3.1**Standartlar ile kalite yönetim sistemi ve projeler arasındaki ilişki**

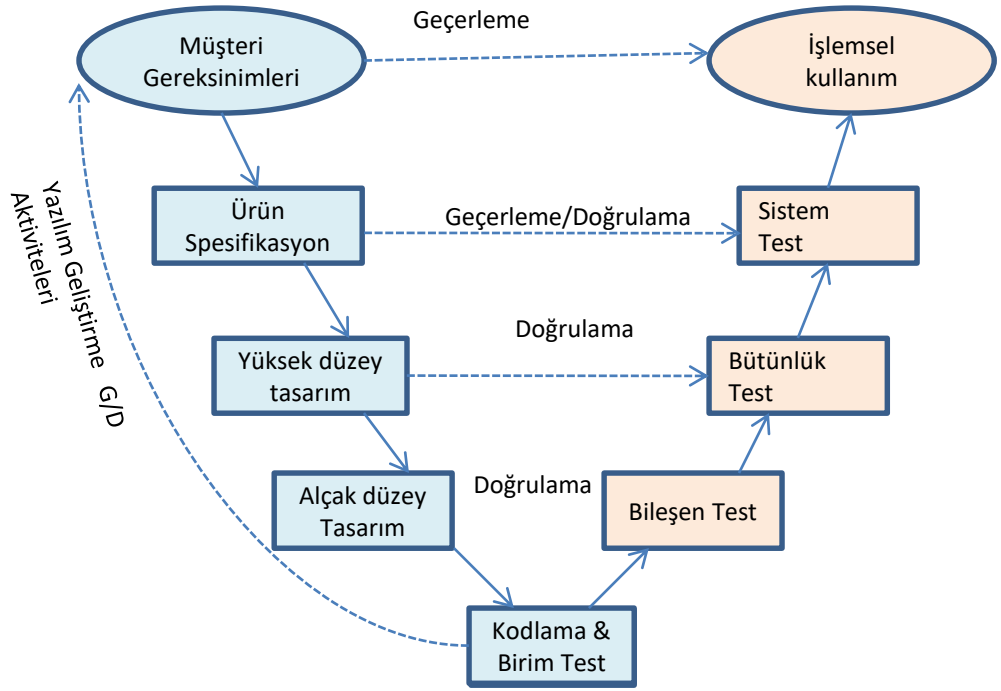
3.2. **Doğrulama ve Geçerleme**

- Doğrulama(verification) ve Geçerleme (validation), yazılım geliştirme süreci adımlarında ürün veya ara ürünlerin istenilen özelliklere uygunluğunu incelemek üzere gerçekleştirilmektedir.
- Geçerleme ile, “ Doğru yazılım üretildi mi?”,
- Doğrulama ile ise ,” Yazılım doğru yolla üretildi mi?”, sorularına cevap aranır.

V model yaklaşımı

- Yazılım geçerleme doğrulama, yazılım geliştirme sürecinde V model yaklaşımı ile gerçekleştirilmektedir(Şekil 3.2).

V Model yaklaşımı



Şekil 3.2 Yazılım geliştirme ve Test sürecindeki Geçerleme & Doğrulama Aktiviteleri (V Model yaklaşımı)

3.3. Gözden Geçirme Ve Onaylama

- Yazılım geliştirme sürecini oluşturan her basamak tamamlanınca, durak noktalarında yapılan işler gözden geçirilmekte ve gerekli düzeltmeler yapılmaktadır

3.3.3. Gözden Geçirme ve İnceleme Tipleri

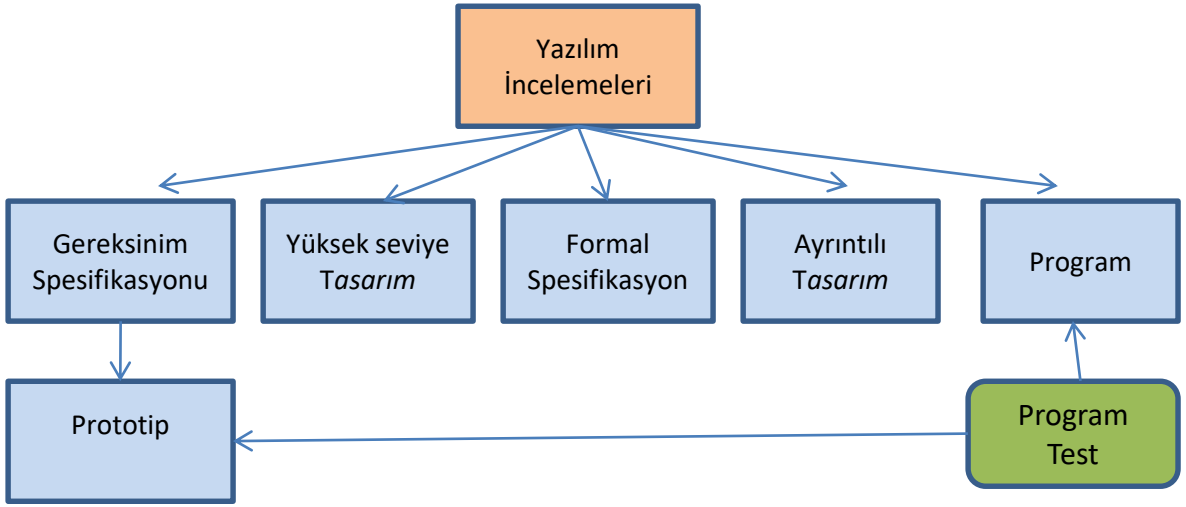
- Gözden Geçirme tipleri,
- formal olmayan gözden geçirme (informal review),
- yapısal denetim (*Walkthrough),*
- Inceleme *(Inspection) ve*
- *round robin peer review* olmak üzere belirtilebilir.

3.3.4. Gözden Geçirme Ve Onaylama Basamakları

Yazılım geliştirme sürecinde gözden geçirme işlemi, genel olarak;

- sistem analizi,
- yazılım geliştirme plânı,
- gereksinim analizi,
- tasarım,
- kodlama,
- sınama,
- bakım ve onarım
- basamaklarının tamamlandığı "durak noktaları"nda (milestone) uygulanmaktadır.

Şekil 4.1. Statik ve Dinamik geçerleme ve Doğrulama



5.1.Yazılım Sınama (Test)

- 5.1.Yazılım Sınama
- Sınama (testing); bir programdaki hataları bulmak amacı ile yapılan işlemlerdir.
- Sınama, yazılımın

  a) fonksiyonel,

  b) performans,

  c) dayanıklık,

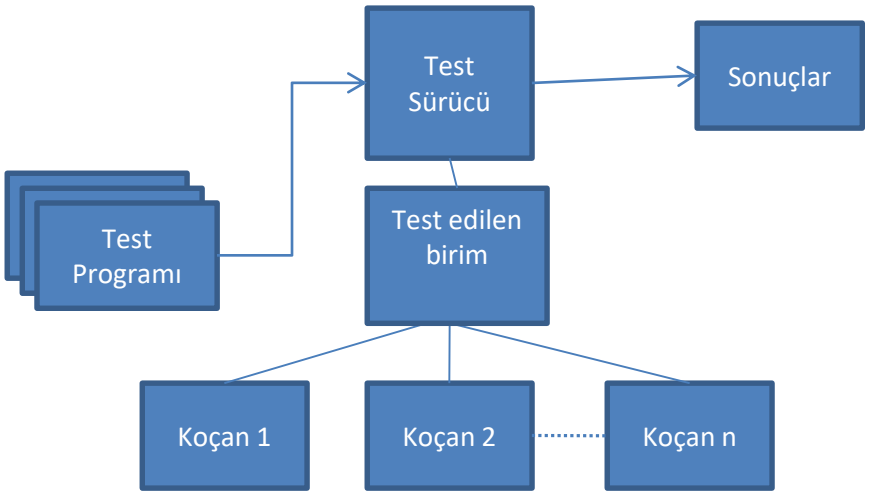
  d) yapısal

bakımlardan yeterliğini denetlemektedir.

5.2.1.Birim Test

- Ünite (birim) testi, yazılım tasarımının en küçük birimi olan modül üzerinde uygulanmaktadır. Ayrıntılı tasarım tanımlarına dayanılarak, modül içerisindeki hataları bulmak üzere, önemli kontrol yolları sınanmaktadır.
- Saydam kutu testi olarak uygulanan bu işlem, çok sayıdaki modül üzerinde, paralel olarak yürütülmektedir.
- Birim testinde; modülün arabirim, lokal veri yapısı, kontrol yapıları arasındaki ana yollar, hata arama yolları ve modül sınırları sınanmaktadır.

Şekil 5.2. Birim Test Ortamı



Birim Test

- Test senaryosu (test case); belirli bir program yolunu işlemek ya da özel bir gereksinime uygunluğu onaylamak amacı ile düzenlenen bir dizi sınama verisinden ve buna ilişkin işlemlerden oluşturulmaktadır.
- Test programlarının geliştirilmesi, diğer yazılımlar gibidir. Geliştirmeye de, test plânı uyarınca ve yazılım tasarımı ile birlikte başlanmalıdır.
- Modülün bağımsız olmaması halinde, sınamada diğer modüller de dikkate alınmalıdır. Bu amaçla her ünite testi için bir “test sürücü”(driver) ve/veya “koçan”(stub) yazılımı geliştirilmektedir.

- Test sürücü (test driver); test programı verisini alarak test edilecek modüle ileten ve test sonucunu yazan bir ara programdır.
- Koçan (stub); bir kukla (dummy) alt proğram olup, test edilen modülün altprogramını temsil etmektedir.

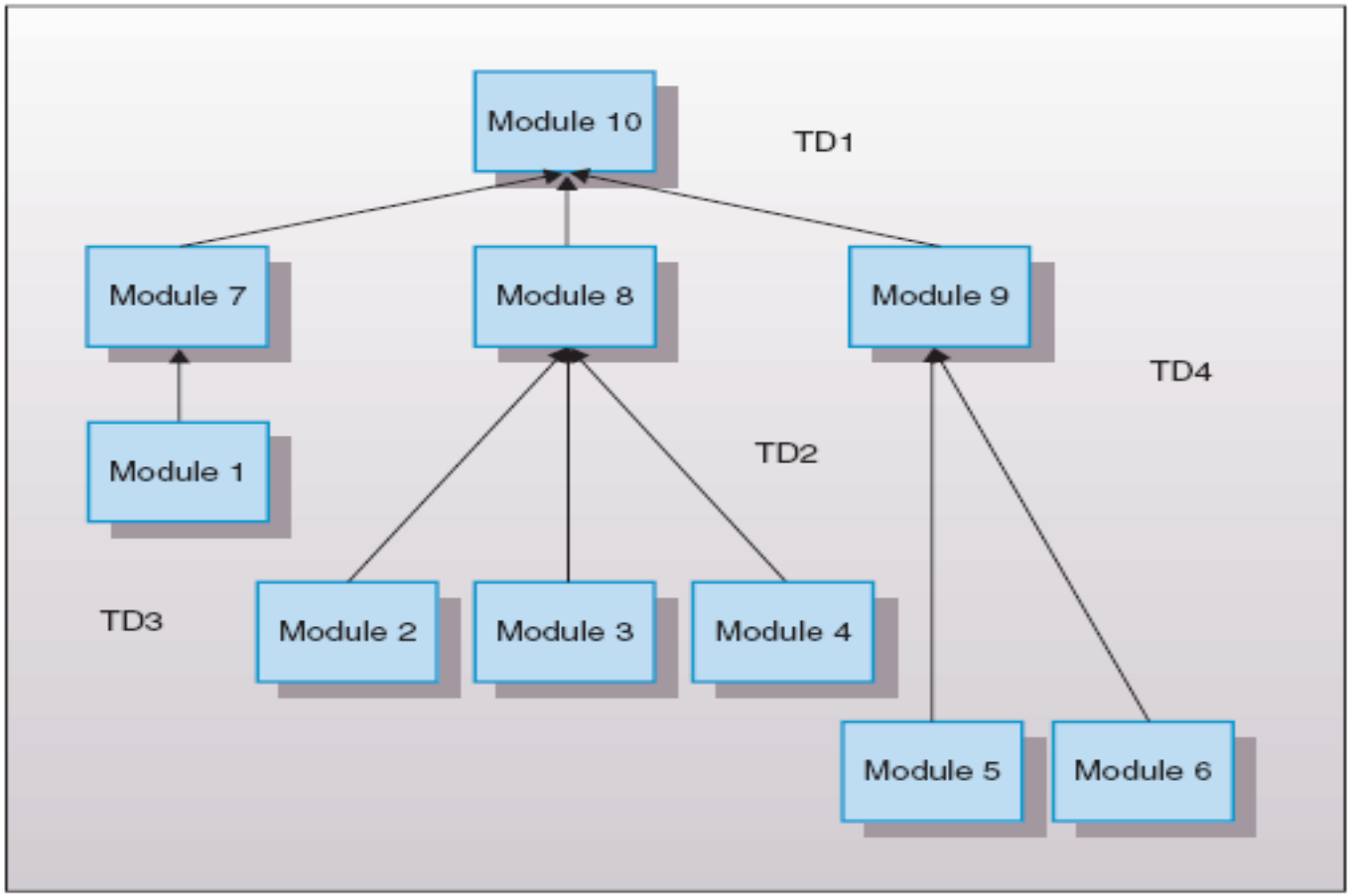
5.2.2.Bütünleme Testi

- Modüller bağımsız olmayıp, birbirilerine bağlı olmalıdır. Bu bağlantı, “yazılım arabirimi” (software interface) ile sağlanmaktadır.
- Modüllerin birleştirilmesi sırasında veri kaybı, dikkatsizlik nedeni ile birbirini ters etkileme, alt fonksiyonların birleştirilmesiyle beklenilen ana fonksiyonunun gerçekleşmemesi, her birinde göze alınabilen hata toleranslarının eklenerek büyümesi, genel veri yapılarının sorun yaratması söz konusudur. Bu hata ve sorunları bulup gidermek için, modüllerin birleştirilerek ana programın oluşturulmasında, bütünleme testi uygulanmalıdır.

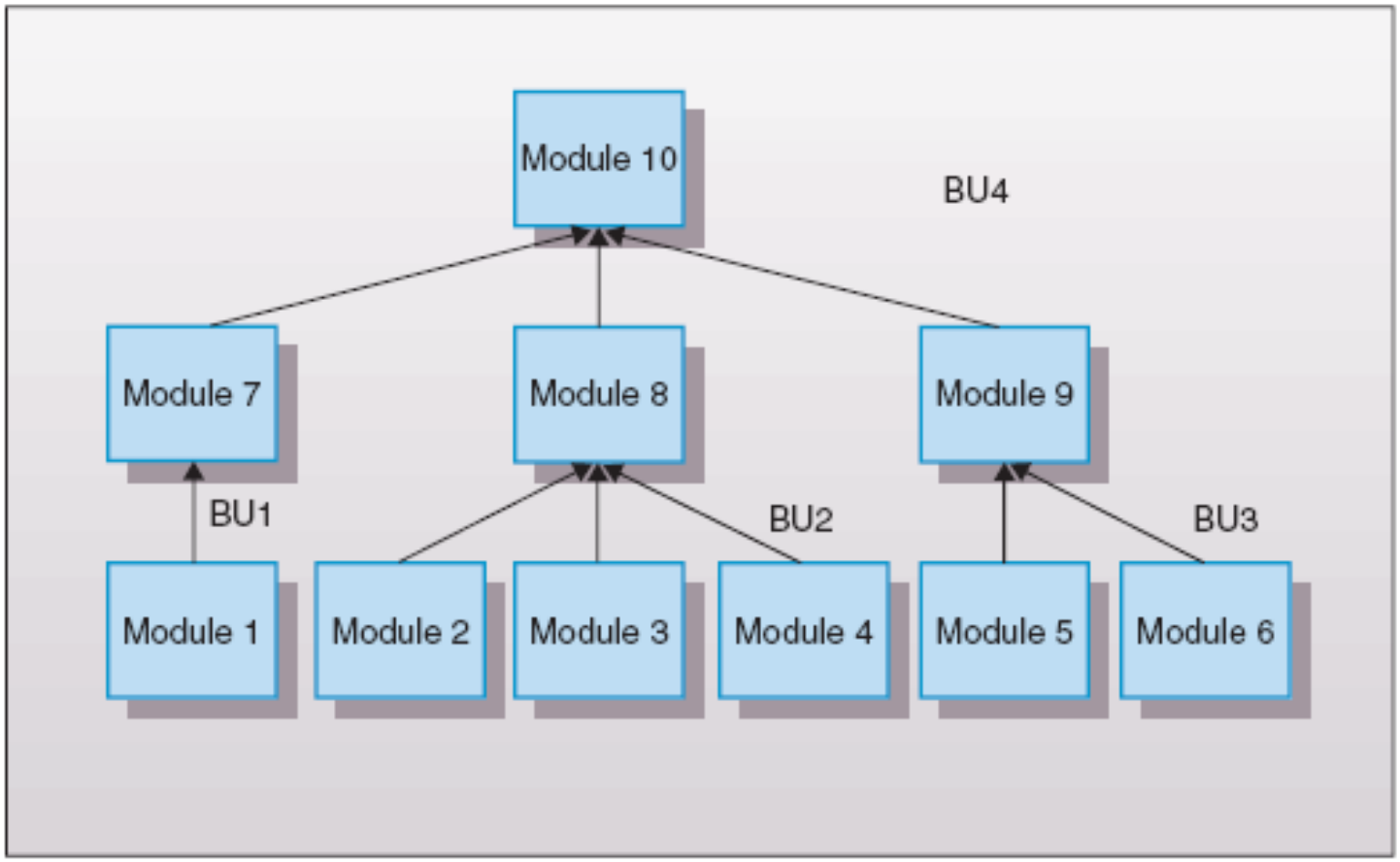
Bütünleme testi

- a) bütün olarak sınama,
- b) artırmalı sınama olarak,
- iki ayrı biçimde birleştirildikten sonra gerçekleştirilmektedir.
- Artırmalı sınamada, modüllere teker teker birbirine bağlanmaktadır.
- Artırmalı sınama,
  - yukarıdan aşağı ve
  - aşağıdan yukarı olarak iki ayrı şekilde uygulanmaktadır.

Şekil 5.3.Yukarıdan Aşağı Bütünleme Testi



Sekil.5.5.Aşağıdan Yukarı Bütünleme Testi



5.2.2.3.Regresyon testi

- **Regresyon Testi** Sınanmış olan bir program veya program parçası üzerinde değişiklik veya ekleme yapılması halinde, tümünün bir kez daha sınanmasıdır.
- Uygulama ortamlarında gerekli değişiklikler ve sabitlemeler yapıldıktan sonra yeniden yapılan testlere regresyon testi denilir.
- Başka bir tanımla, Regresyon Testi, önceden test edilmiş bir yazılımın çeşitli değişiklerden geçtikten sonra da hatasız bir şekilde çalışmasını sağlamak amacıyla yeniden test edilmesi işlemidir.

5.2.3.Sistem Testi

- Bilgisayar sistemi, donanım ve yazılım alt sistemlerinden oluşmaktadır. Bu nedenle, yazılım alt sisteminin kendi başına sınanması yeterli olmayıp, bilgisayar sistemi içerisinde de denetlenmelidir.
- Sistem testinin amacı; sistemin bütün öğelerinin uygun olarak bir araya getirildiğinin ve her birinin işlevini tam olarak gerçekleştirebildiğinin onaylanmasıdır.
- sistem testi;
  - a) düzeltme testi, b) güvenlik testi,
  - c) dayanıklılık testi, d) yetenek testi
- biçimlerinde uygulanmaktadır.

Sistem Testi

- .**Güvenlik testi:** sistemin zararlı dış müdahalelerden ve bilgi hırsızlığından korunabildiğinin kanıtlanmasıdır.
- **.Dayanıklık (stres) testi;** sistemin miktar, frekans ya da hacım bakımından anormal biçimde yüklenmesi hallerindeki dayanıklığını ölçmek amacı ile düzenlenmektedir.
- **Yetenek (performance) testi**; gerçek zamanlı ve gömülü sistemlerde, yazılım işlem süresinin bilgisayara dayalı sistem ile uyarlığını sınamaktadır. Yeteneğin sınanması, her test basamağında uygulanmaktadır.

5.2.4. Onaylama Testi

- Onaylama Testi, Bütünleme testi sonunda, yazılım bir paket halinde derlenmiş, arabirim hataları bulunmuş ve düzeltilmiş olmaktadır. Bundan sonra, *onaylama testi* yapılmaktadır. Bu testte, yazılımın müşteri ve kullanıcı beklentilerini gerçekleştirme olanağı denetlenmektedir.
- Bu amaçla; onaylama testi, düzenlik testi ve kabul muayenesi olarak yürütülmektedir

Test Tipleri

Fonksiyonel-performans ve dayanıklık testlerine, sistemin dış spesifikasyonlarına ve gereksinimlerine dayandırıldığı için, **kara kutu testi** (black box testing) adı verilmektedir.

- Buna karşılık, yapısal denetimde modül düzeyinde programın deyimleri ya da dalları sınanarak iç yapısı incelenmektedir. Bu şekilde uygulanan sınama yöntemine de **saydam kutu testi** (white box, glass box testing) denilmektedir.

Saydam kutu testi

Saydam kutu testinde, işlemsel (procedural) tasarımın kontrol yapısı kullanılmaktadır. Bu test ile;

- Bir modüldeki bütün bağımsız yolların en az bir kez çalışacağı garanti edilmekte,
- Bütün mantıksal kararların "doğru" ve "yanlış" durumları denenmiş olmakta,
- Bütün döngülerin kendi içinde ve çevresinde işlerliği sağlanmakta,
- İç veri yapıları denenerek, geçerliliği güvence altına alınmaktadır.
- Saydam kutu testinin uygulanmasında, temel yol testi ve döngü testi teknikleri kullanılmaktadır.

Saydam Kutu: Temel Yollar Testi

- Temel yollar testi, işlemsel tasarımın mantıksal karmaşıklığını ölçmek ve bu ölçüye göre uygulama yolları için bir temel grup oluşturmak esasına dayanmaktadır.
- Test programları, test sırasında programdaki her deyimi en az bir kez uygulayarak denemektedir.

Uygulama;

- Ayrıntılı tasarım veya kaynak programa dayanarak, bir akış grafı çizmek .
- Akış grafı üzerinde döngüsel karmaşıklık (McCABE) ölçüsünü saptamak,
- Doğrusal bağımsız yolların temel grubunu ve düğümleri belirlemek,
- Temel grubun içerdiği her yolun denenmesi için birer test programı düzenlemek,
- Her test programını uygulamak ve beklenen sonuç ile karşılaştırmak

şeklinde, basamaklar halinde yürütülmektedir

Saydam Kutu:Döngü Testi

- Döngü testi; bir saydam kutu testi olup, temel yollar analizine ek olarak yürütülmektedir. Bir döngü içerisinde bir giriş ve bir çıkışlı olarak soyutlanmış bulunan yollar da, birer döngü halinde ayrıca sınanmaktadır. Döngü testinin amacı; döngü içerisindeki başlama hatalarının, indeksleme ve artırma hatalarının, döngüyü sınırlama hatalarının bulunmasıdır.
- Test sonunda, döngü yapısının geçerliği onaylanmış olmaktadır.

Kara Kutu Testi

- Kara kutu testi; yazılımın bütünlenmesi sırasında uygulanan ve yazılım arabirimi üzerinde yapılan bir sınamadır. Bu sınama ile, yazılım işlevlerinin yerine getirildiği, girdilerin kabul edildiği, çıktıların doğru olarak bütünlüğün sağlandığı gösterilmektedir. Kara kutu testinde yazılımın mantıksal iç yapısından çok, temel sistem modeli denenmiş olmaktadır. Bu nedenle, kara ve saydam kutu testleri birlikte uygulanarak, yazılım arabiriminin geçerliği onaylanmakta ve yazılımın iç işlerliğinin doğruluğu kısmen güvence altına alınmaktadır

7.1. Test Yönetimi

- Test kapsamında gerçekleştirilen işlemler (Şekil 7.1) de görüldüğü gibi planlama, tasarım ve gerçekleştirme adımları ile yürütülmektedir.
- Yazılım testlerini tanımlamak, planlamak, düzenlemek ve belgelemek için, **test spesifikasyonu** adı verilen bir belge düzenlenmektedir. Bu belge, genel hatları ile;
- Test plânları: test şekilleri, zamanlama, gider, ortam ve kaynaklar
- Test senaryoları
- Test işlemleri: her testin tanımlanması (bütünleme biçimi, amacı ve test edilen modüller, özel araç ve teknikler, gideri, test programı verisi) ve beklenen sonuçlar
- Gerçek test sonuçları
- Referans
- Ekler

Yazılım ölçümü

- Yazılım ölçümü zordur:
- • yorumlama engeli yüksektir.
- • Zorluğun nedenleri:
  - • Yazılımın karmaşıklığı
  - • Ölçütlerin nicel doğası
- • Yazılımı neden ölçeriz?
- • Ne kadar iyi bir ürün ortaya çıkardığımızı anlamak
  - • Ne kadar iş yapacağımızı kestirmek
  - • Böylece ne kadar zaman ve para harcayacağımızı anlamak
- • Ölçülemeyen ilerleme yönetilemez:

YAZILIM KALİTE ÖLÇÜTLERİ

- Nicel kalite ölçütleri farklı kişilerce farklı şekillerde öbeklenmekte ve farklı dallara ayrılmaktadır.
- ISO 9126 kalite ölçütleri: • İşlevsellik • Uygunluk, doğruluk, güvenlik, … • Güvenilirlik • Olgunluk, hata bağışıklığı, … • Kullanılabilirlik • … • Verimlilik/Etkinlik • … • Bakım kolaylığı • … • Taşınabilirlik • …
- McCall ve arkadaşlarının kalite ölçütleri: • İşlevsel ölçütler • Doğruluk, Güvenilirlik, Bütünlük, Kullanılabilirlik, Verimlilik • Değiştirilme ölçütleri • Bakım kolaylığı, Esneklik, Sınanabilirlik • Taşınma ölçütleri • Taşınabilirlik, Yeniden Kullanılabilirlik, Birlikte Çalışabilirlik
- McConnell'a göre kalite ölçütleri: • İç kalite ölçütleri • Dış kalite ölçütleri

Nesneye yönelik ölçütler

- Nesneye yönelik ölçütler:
- • Kaliteli bir yazılıma götüren tasarım ilkelerine yöneliktirler. • NYP'de çözümleme ve tasarım arasında kopukluk olmadığı için, aynı ölçütler çözümleme ve kodlama aşamalarında da kullanılabilir.
- • Chidamber ve Kemerer'in ölçütleri (CK metrics suite): • WMC: Sınıftaki ağırlıklı metot sayısı (Weighted Methods per Class).
- • DIT: Kalıtım ağacının derinliği (Depth of Inheritance Tree). • NOC: Alt sınıf sayısı (Number of Children)
- • RFC: Sınıfın yanıt kümesinin eleman sayısı (Response For a Class)
- • CBO: Sınıflar arası bağlaşım (Coupling Between Objects)
- • LCOM: Uyum eksikliği (Lack of COhesion in Methods)

Süreç İyileştirme

- YAZILIM GELİŞTİRME ORTAMI SERTİFİKASİSTEMLERİ
- CMM
- CMMI
- SPICE
- ISO

8.1. CMMI

- Yazılım sektöründe ürünün kalitesini ağırlıkla onu üreten sürecin kalitesi belirlemektedir ve süreç odaklı kalite yaklaşımı vardır.
- CMMI etkin bir yazılım sürecinin anahtar elemanlarını tanımlayan bir çerçeve model ve yazılım sürecinin olgunluğunu değerlendirmek için bir ölçüm aracıdır.
-

CMMI Yeterlilik Düzeyleri

- CMMI modeli, süreçler için beş yeterlilik düzeyi tanımlamaktadır.
- Başlangıç
- Yönetilebilir
- Tanımlı
- Sayısal Yönetilebilir
- Eniyilenebilir

CMMI Anahtar Süreç Alanları

- Etkin bir yazılım süreci geliştirmek için tanımlanan anahtar süreç alanlarının düzeylere göre dağılımı belirtilecektir. Birinci düzeyde anahtar süreç alanı yoktur.
- ***II. Düzey (Yönetilen)***
  - Gereksinimlerin Yönetimi
  - Proje Planlama
  - Proje İzleme ve Kontrol
  - Tedarikçi Sözleşme Yönetimi
  - Ölçümleme ve Analiz
  - Süreç ve Ürün Kalite Güvence
  - Konfigürasyon Yönetimi

CMMI Anahtar Süreç Alanları

- ***III. Düzey (Tanımlı)***
  - Gereksinimleri Geliştirme
  - Teknik çözüm
  - Ürün entegrasyonu
  - Doğrulama Onaylama
  - Kurumsal süreç tanımlama ve odaklanma
  - Kurumsal eğitim
  - IPPD (entgre ürün ve süreç geliştirme) için tümleşik proje yönetimi
  - Risk yönetimi
  - Entegrasyon için organizasyonel altyapı

CMMI Anahtar Süreç Alanları

- ***IV. Düzey (Niceliksel Yönetilen)***
  - Organizasyonel süreç başarımı
  - Niceliksel proje yönetimi

- ***V. Düzey (eniyileştirilmiş)***
  - Organizasyonel yenilenme ve yaygınlaştırma
  - Nedensel analizler ve çözümleme

P-CMM

- P-CMM temel hedefleri;
- Yazılım geliştirme yeteneğini tanımlamak ve sağlamak
- iş gücünün yeteneklerini artırarak organizasyonun yeteneklerini artırmak ve iyileştirmek,
- Her bir çalışanın ile organizasyonun hedeflerini birlikte tutmak
- İnsan değerlerini elinde tutmaktır.